BootLoader ISO Image File To Edit Hex

씨익

소개글

헥사 코드만을 편집 만을 이용해서 Boot-Loader를 만들고, 동시에 Bootable ISO Image 파일을 만들어보는 것이다.

목차

Make Boot-Loader without compiler

Make Boot-Loader with some tools

Information

Subject	Make Boot-Loader with some tools
Author	siik iticworld@hanmail.net
Version	0.1
Summary	"Boot-Loader" 를 컴파일러 없이 만드는 프로젝트 중에서 개념을 잡기 위해, 공개된 자료를 이용해서 iso bootable image를 만드는 것이다. 이용한 툴은 nasm, nero burning 이며, 관련한 소스는 http://www.viralpatel.net/taj/tutorial/hello_world_bootloader.php 페이지에서 얻었다.

Histroy

Version	Author	Description
0.1	siik iticworld@hanamil.net	문서 최초 작성

Milestones

1. "ITICWORLD" 란 단어를 Pixel 단위로 직접 컨트롤 하여 출력하는 Boot Image를 컴파일러 없이 Hex Editor 만 이용해서 만드는 것이다.
2. ISO Image 파일 포맷 구조를 이용해서, Hex Editor로만 ISO Bootable CD를 만드는 것이다.

Introduction

"Make Boot-Loader without compiler" 란 프로젝트를 진행하기 위한 초기 프로젝트로, 공개된 소스와 일반적으로 가용할 수 있는 툴들을 이용해서 iso image file에 만든 boot image를 boot 가능하도록 복사하는 것이다.

Prerequisites

1. Nero burning ROM

Nero Burning ROM은 CD, DVD, 및 Blue-ray 디스크를 Recording 및 복사하며, 스크래치, 노화 또는 열화에 관계 없이

Recorded Contents 의 신뢰성을 보장해준다. 이 소프트웨어의 기능 중에서 Bootable CD를 만들 수 있는 기능이 존재하고, 이를 통해서 만든 boot image를 bootable cd를 실행할 때, 이 바이너리 코드를 실행할 수 있도록 하는데, 이 소프트웨어가 사용된다.

2. NASM

Net-wide Assembler, NASM은 80x86 과 80x64 어셈블러다. 이를 통해서 다양한 objec파일을 만들 수 있는데, boot image 역시 만들 수 있다. 이는 boot image 를 만들기 위한 Assembly Code를 binary code 로 컴파일하는데, 사용된다.

3. VirtualBox

VirtualBox는 일반적인 목적의 x86 하드웨어의 가상화를 위한 소프트웨어다. 이 소프트웨어를 통해서 만든 부팅가능한 iso image 파일을 실제 작동하는지 테스트 할 수 있다.

Description

1. Assembly Source

```
[BITS 16]                           ;Tells the assembler that its a 16 bit code
[ORG 0x7C00]                        ;Origin, tell the assembler that where the code will
                                    ;be in memory after it is been loaded
MOV AL, 65
CALL PrintCharacter
JMP $                               ;Infinite loop, hang it here.



PrintCharacter:                     ;Procedure to print character on screen
                                    ;Assume that ASCII value is in register AL
MOV AH, 0x0E                        ;Tell BIOS that we need to print one charater on screen.
MOV BH, 0x00                        ;Page no.
MOV BL, 0x07                        ;Text attribute 0x07 is lightgrey font on black background

INT 0x10                            ;Call video interrupt
RET                                 ;Return to calling procedure

TIMES 510 - ($ - $$) db 0           ;Fill the rest of sector with 0
DW 0xAA55                           ;Add boot signature at the end of bootloader
```

위의 소스는 http://www.viralpatel.net/taj/tutorial/hello_world_bootloader.php에서 발췌하였다.

2. Compile to make boot binary file

```
nasm iticworld.boot.loader.asm -f bin -o boot.bin
```

-f 는 출력 파일 포맷을 지정하는 옵션이다.

-o 는 출력 파일 이름을 지정하는 옵션이다.

“binary 파일 포맷으로 boot.bin을 만들어라.” 라는 명령문이다.

3. Bootable CD Using Nero

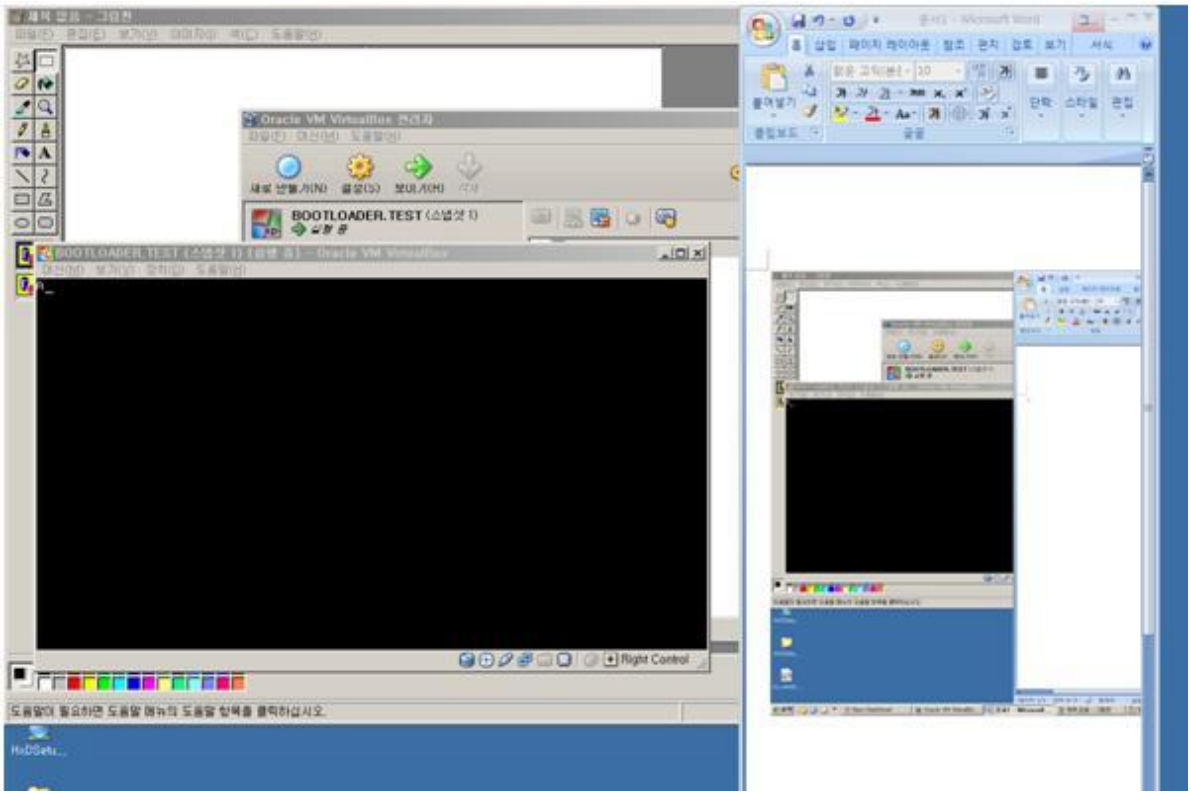
부팅 이미지 데이터를 선택해서 부팅 CD를 만들 수 있다. 이를 통해서 image.iso파일을 만들었다.

4. Test Using VirtualBox

Windows XP 용 가상 머신을 만들었고, CD-ROM으로 부팅을 할 수 있도록 옵션을 설정하였다. 그리고 이를 통해서 실제 테스트해 보았다.

5. Result

아래의 이미지는 VirtualBox를 통해서 'A' 를 출력한 스크린샷이다. 희미하지만, 정확히 출력되었고, 그 이후 "JMP $" 를 수행해서 무한히 반복중인 모습이다.



Conclusion

이번 프로젝트의 목적은 http://www.viralpatel.net/taj/tutorial/hello_world_bootloader.php를 통해서 간단한 Boot-Loader를 직접 만들어보고, 이를 테스트할 수 있는 환경을 구축하고 이 또한 테스트하는 것이다. 실제로의 목적은 가용한 Tool이 존재하지 않을 때, Hex Editor 만 가지고, Boot ISO Image 파일을 만드는 것이다. 하드 디스크에 직접 쓰고 싶지만, 일단 하드디스크의 파티션 Specification 까지 알아야 하는 문제가 있어 보여서, 진행에 많은 시간이 걸릴 것 같아서, 일단 ISO 이미지로 국한 시킨 것이다. 앞으로 진행해야 할 것은,

1. "ITICWORLD" 란 단어를 Pixel 단위로 직접 컨트롤 하여 출력하는 Boot Image를 컴파일러 없이 Hex Editor 만 이용해서 만드는 것이다.

2. ISO Image 파일 포맷 구조를 이용해서, Hex Editor로만 ISO Bootable CD를 만드는 것이다.

Reference

http://www.nero.com/kor/products.html

http://www.nasm.us/xdoc/2.09.10/html/nasmdoc1.html#section-1.1

https://www.virtualbox.org/wiki/VirtualBox

http://www.viralpatel.net/taj/tutorial/hello_world_bootloader.php

Resource

Image.iso

[Make Boot-Loader without compiler] Make Boot-Loader with .docx

boot.bin

DEVELOPMENT

Make Bootable ISO Image File only editing hex code

Information

Subject	Make Bootable ISO Image File only editing hex code
Version	0.2
Author	siik iticworld@hanmail.net
Summary	Make Bootable ISO Image File only editing hex code Build a Boot-Loader to Print 'ITICWORLD', and Analysis it.
SVN	https://iticworld.net:7281/svn/MAKE.BOOT.LOADER.WITHOUT.COMPILER/Make Bootable ISO Image File only editing hex code/

History

Version	Date	Author	Description
0.1	201202162210	siik iticworld@hanmail.net	Make Bootable ISO Image File only editing hex code Build a Boot-Loader to Print 'ITICWORLD', and Analysis it.
0.2	201202162414	siik iticworld@hanmail.net	3. Make Bootable ISO Image Source 정리 Milestones 추가: Byte Align 조정을 이용해서, 각각 어울리는 타입으로 변경할 것

Introduction

hex code만으로 Bootable ISO Image 파일을 만들 것이다. 그를 위해서 우선 이미 만들어진 image.iso 파일을 분석할 필요가 있다. 또한 ISO9660과 관련하여 정보를 수집할 필요가 있다.

Prerequisites

image.iso

VirtualBox

Milestones

ISO9660 요약

El Torito Bootable CD-ROM Format Specification Version 1.0 January 25, 1995 요약

Volume and File Structure of CDROM for Information Interchange(EMCA 119) 요약

Virtual CD-ROM 제작

ISO Image File System Parser Library

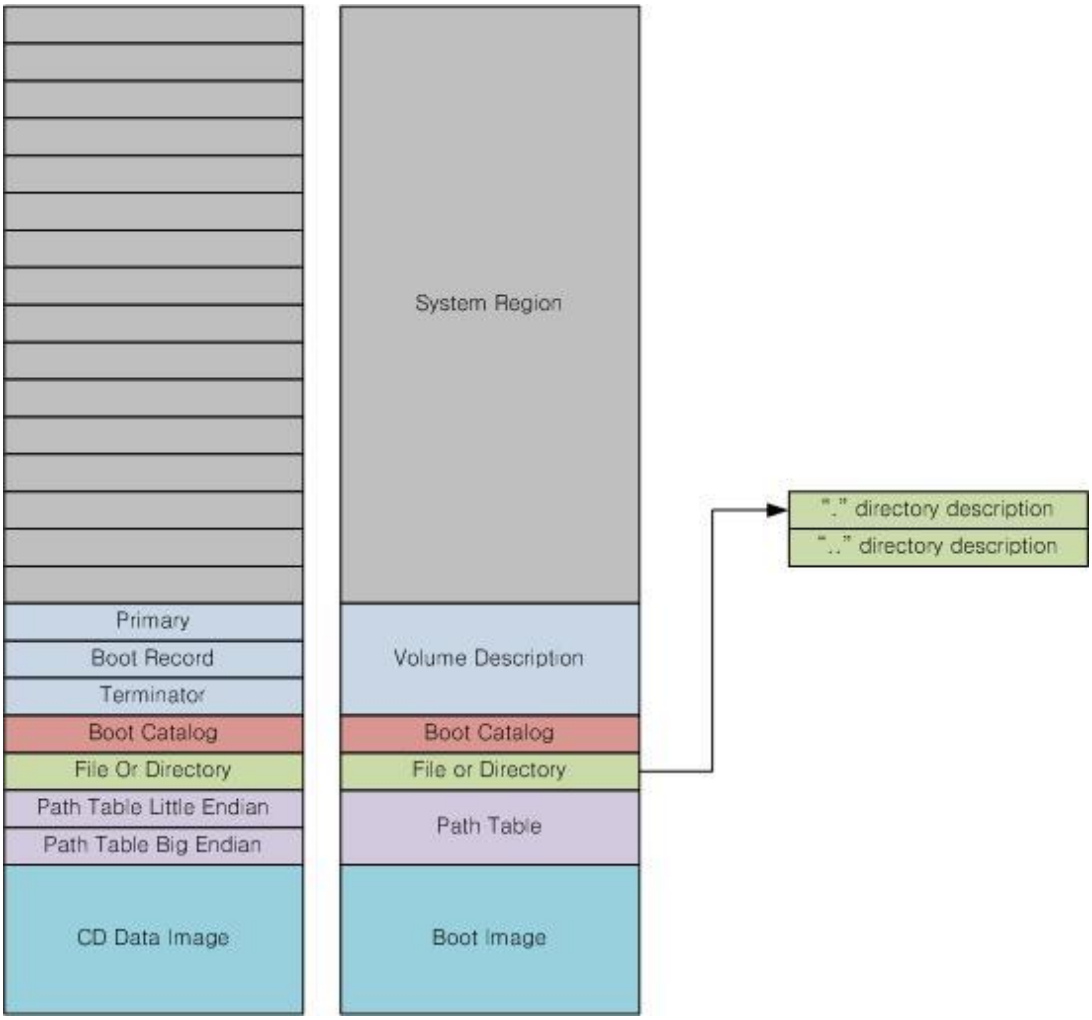
Nero Burning Rom을 이용하여 Bootable ISO Image을 만들 때, 최소 크기가 아닌 600*2048크기로 이미지 파일이 만들어지는 정확한 이유를 알아야 한다.

#pragma pack(n)을 이용해서, Byte Align을 조정하고, unsigned char 형으로 정의한 데이터형을 구조체로 정의하여 반영할 것

Description

1. image.iso volume diagram

한 섹터의 기본적인 크기는 2048 bytes이다. 한 섹터의 크기는 2048이거나 혹은 2^n의 크기를 가질 수 있지만, 2048이 범용적으로 사용된다. Boot Catalog 와 Boot Image 영역은 "El Torito Bootable CD-ROM Format Specification Version 1.0 January 25, 1995" 에 잘 묘사되어 있다. 그림에서 보듯이 image.iso는 System Region, Volume Descriptor, Boot Catalog, Directory Description, Path Table, Boot Image 순으로 구성되어 있다. Volume Descriptor는 Primary Volume Descriptor, Boot Record Volume Descriptor, 그리고 Volume Terminator으로 구성되어 있다. Path Table의 경우 같은 정보를 little endian, big endian로 기술하고 있는데, 각 Path Table은 한 섹터를 차지하고 있다. 각각 영역에 들어가야 할 정보들은, Volume and File Structure of CDROM for Information Interchange(EMCA 119)에 묘사되어 있고, image.iso에서 사용하는 것은 Primary Volume Descriptor, Boot Record Volume Descriptor, Set Volume Terminator Description, Path Table, Directory Description, 등이다. 또, Boot Catalog관련해서는 Validation Entry, Initial/Default Entry가 "El Torito Bootable CD-ROM Format Specification Version 1.0 January 25, 1995" 에 묘사되어 있다. 이와 같은 구조는 Multi-Booting을 지원하지 않고, Single Booting을 지원하는 구조다. Multi-Booting에 대해서는 "El Torito Bootable CD-ROM Format Specification Version 1.0 January 25, 1995" 을 참고하면 된다.



2. Analysis image.iso file

```
locationOfFirstSectorOfBootCatalog : 0x00000013
= SET VOLUME TERMINATOR DESCRIPTORS ================
= BOOT CATALOG =====================================
bootCatalogValidation->headerID : 1
bootCatalogValidation->platformID : 0
bootCatalogValidation->IDString : NERO BURNING ROM
bootCatalogValidation->checksum : 138
bootCatalogValidation->keyByte1 : 0x55
bootCatalogValidation->keyByte2 : 0xAA
initialEntry->bootIdicator : 0x88
initialEntry->bootMediaType : 2
initialEntry->loadSegment : 07C0
initialEntry->systemType : 6
initialEntry->sectorCount : 1
initialEntry->loadRBA : 0x00000017
```

```
= DIRECTORY 1 & 2 ================================
- DIRECTORY 1 ------------------------------------
extentDirDesc->LengthOfDirectoryRecord : 34
extentDirDesc->extendedAttributeRecordLength : 0
extentDirDesc->locationOfExtent : 20
extentDirDesc->dataLength : 2048
directoryRecordForRootDirectory.recordingDateAndTime : 20122141410
extentDirDesc->fileFlags : 2
extentDirDesc->fileUnitSize : 0
extentDirDesc->interleaveGapSize : 0
extentDirDesc->volumeSequenceNumber : 1
extentDirDesc->lengthOfFileID : 1
directoryRecordForRootDirectory.fileID : 0x00
- DIRECTORY 2 ------------------------------------
extentDirDesc->LengthOfDirectoryRecord : 34
extentDirDesc->extendedAttributeRecordLength : 0
extentDirDesc->locationOfExtent : 20
extentDirDesc->dataLength : 2048
directoryRecordForRootDirectory.recordingDateAndTime : 20122141410
extentDirDesc->fileFlags : 2
extentDirDesc->fileUnitSize : 0
extentDirDesc->interleaveGapSize : 0
extentDirDesc->volumeSequenceNumber : 1
extentDirDesc->lengthOfFileID : 1
directoryRecordForRootDirectory.fileID : 0x01
= PATH TABLE =====================================
pathTable->lengthOfDirID : 1
pathTable->extendedAttributeRecordLength : 0
pathTable->locationOfExtent : 0x00000014
pathTable->parentDirectoryNumber : 0x00000001
pathTable->directoryID :
```

3. Make Bootable ISO Image

```
PRIMARY_VOLUME_DESCRIPTOR primaryVolumeDesc={
0x01, // PRIMARY VOLUME DESCRIPTOR NUMBER
{0x43,0x44,0x30,0x30,0x31}, // STANDARD ID "CD001"
0x01, // VERSION 1
0x00, // UNUSED
"", // SYSTEM ID
"SIIK BOOTABLE CD", // VOLUME ID
```

```
{0,}, // UNUSED
{0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x18}, // VOLUME SPACE SIZE, {little endian:4, big endian:4}, 24=0x18
{0,}, // UNUSED
{0x01,0x00,0x00,0x01}, // VOLUME SET SIZE, {little endian:2, big endian:2}
{0x01,0x00,0x00,0x01}, // VOLUME SEQUENCE NUMBER, {little endian:2, big endian:2}
{0x00,0x08,0x08,0x00}, // LOGICAL BLOCK SIZE, {little endian:2, big endian:2}
{0x0A,0x00,0x00,0x00,0x00,0x00,0x00,0x0A}, // PATH TABLE SIZE, {little endian:4, big endian:4}
{0x15,0x00,0x00,0x00}, // PATH TABLE SECTOR, {little endian:4}
{0x00,0x00,0x00,0x00}, // OPTIONAL PATH TABLE SECTOR, {little endian:4}
{0x00,0x00,0x00,0x16}, // PATH TABLE SECTOR, {big endian:4}
{0x00,0x00,0x00,0x00}, // OPTIONAL PATH TABLE SECTOR, {big endian:4}
{                 // ROOT DIRECTORY DESCRIPTOR
            22, // DIRECTORY DESCRIPTORS SIZE
            0, // EXTENDED ATTRIBUTE RECORD LENGTH
            0x14,0x00,0x00,0x00,0x00,0x00,0x00,0x14, // EXTENT LOCATION, {little endian:4, big endian:4}
            0x00,0x08,0x00,0x00,0x00,0x00,0x08,0x00, // DATA LENGTH
            112,2,16,16,1,0,0, // YEAR,MONTH,DAY,HOUR,MINUTE,SECOND,GMT
            2, // FILE FLAGS
            0, // FILE UNIT SIZE
            0, // INTERLEAVE GAP SIZE
            0x01,0x00,0x00,0x01, // VOLUME SEQUENCE NUMBER {little endian:2,big endian:2}
            0x01, // ID STRING LENGTH
            0x00 // ID, "."
},
"", // VOLUME SET ID
"siik", // PUBLISHER ID
"siik", // DATA PREPARER ID
"hex editor", // APPLICATION ID
"", // COPYRIGHT FILE ID
"", // ABSTRACT FILE ID
"", // BIBLIOGRAPHIC FILE ID
"201202161601000", // CREATION DATE
"201202161601000", // MODIFICATION DATE
"", // EXPIRE DATE
"", // EFFECTIVE DATE
1, // FILE STRUCTURE VERSION
{0,}, // APPLICATION USE
{0,} // UNUSED
};

BOOTRECORD_VOLUME_DESCRIPTOR bootRecordVolumeDesc={
0, // VOLUME DESCRIPTOR : BOOT RECORD VOLUME DESCRIPTOR
```

```
{0x43,0x44,0x30,0x30,0x31}, // CD001
1, // VERSION
"EL TORITO SPECIFICATION", // "EL TORITO SPECIFICATION"
{0,}, // UNUSED
{0x13,0x00,0x00,0x00}, // BOOT CATALOG SECTOR NUMBER
{0,} // UNUSED
};

TERMINATOR_VOLUME_DESCRIPTION volumeDescriptorTerminator={
0xFF, // SET VOLUME TERMINATOR
{0x43,0x44,0x30,0x30,0x31}, // CD001
1, // VERSION
{0,} // UNUSED
};

BOOT_CATALOG bootCatalog={
0x01, // VALIDATION HEADER ID
0x00, // VALIDATION PLATFORM ID 0=80x86
{0,}, // VALIDATION RESERVED
"HEX EDITOR", // VALIDATION ID STRING (0x48,0x45,0x58,0x20,0x45,0x44,0x49,0x54,0x4F,0x52)
{0x2D,0x05}, // VALIDATION CHECKSUM
// (0x0001 + 0x4548 + 0x2058+ 0x4445+ 0x5449+ 0x524F+0xAA55 = 0x1FAD3 + 0x052D)
0x55, // CHECK BYTE
0xAA, // CHECK BYTE
0x88, // INTIAL ENTRY IS BOOTABLE
0x02, // EMULATION TYPE : 2 IS 1.44 meg diskette
{0xC0,0x07}, // LOAD SEGMENT
6, // SYSTEM TYPE
0, // UNUSED
{1,0}, // SECTOR COUNT
{0x17,0,0,0}, // RELATIVE LOGICAL BLOCK ADDRESS
{0,}, // UNUSED
};

ROOT_DIRECTORY_DESC ExtentAboutrootDirDesc={
// ROOT "." DIRECTORY DESCRIPTION
0x22, // RECORDED LENGTH
0x00, // EXTENDED DATA LENGHT
{0x14,0x00,0x00,0x00,0x00,0x00,0x00,0x14}, // SECTOR ADDRESS {little endian:4, big endian:4}
{0x00,0x08,0x00,0x00,0x00,0x00,0x08,0x00}, // DATA LENGTH {little endian:4, big endian:4}
{112,2,16,16,1,0,0},              // RECORDING TIME
0x02, // DIRECTORY FILE FLAG
```

```
0x00, // FILE UNIT SIZE
0x00, // INTERLEAVE GAP SIZE
{0x01,0x00,0x00,0x01}, // VOLUME SEQUENCE {little endian:2,big endian:2}
0x01, // FILE ID LENGTH
0x00, // "." FILE ID
// ROOT "." DIRECTORY DESCRIPTION
0x22, // RECORDED LENGTH
0x00, // EXTENDED DATA LENGHT
{0x14,0x00,0x00,0x00,0x00,0x00,0x00,0x14}, // SECTOR ADDRESS {little endian:4, big endian:4}
{0x00,0x08,0x00,0x00,0x00,0x00,0x08,0x00}, // DATA LENGTH {little endian:4, big endian:4}
{112,2,16,16,1,0,0},              // RECORDING TIME
0x02, // DIRECTORY FILE FLAG
0x00, // FILE UNIT SIZE
0x00, // INTERLEAVE GAP SIZE
{0x01,0x00,0x00,0x01}, // VOLUME SEQUENCE {little endian:2,big endian:2}
0x01, // FILE ID LENGTH
0x00, // ".." FILE ID
{0,}
};

PATH_TALBE littleEndianPathTable={
0x01, // IS ROOT DIR
0x00, // EXTEND ATTRIBUTE DATA LENGTH
{0x14,0x00,0x00,0x00}, // LITTLE ENDIAN SECTOR ADDRESS
{0x01,0x00}, // LITTLE ENDIAN SECTOR COUNT
{0,} // UNUSED
};

PATH_TALBE bigEndianPathTable={
0x01, // ROOT DIR
0x00, // EXTEND ATTRIBUTE DATA LENGTH
{0x00,0x00,0x00,0x14}, // BIG ENDIAN SECTOR ADDRESS
{0x00,0x01}, // BIG ENDIAN SECTOR COUNT
{0,} // UNUSED
};

BOOT_IMAGE bootImage={
{0xB0,0x41,0xE8,0x02,0x00,0xEB,0xFE,0xB4,0x0E,0xB7,0x00,0xB3,0x07,0xCD,0x10,0xC3,0x00,},
// BOOT IMAGE boot.bin
{0x55,0xAA}, // CHECK BYTES
{0,} // UNUSED
};
```

위의 코드는 이전에 만들어 놓았던 'A' 만을 출력하는 boot.bin의 바이너리 코드를 이용해서 각각의 데이터 구조에 정보를 삽입하고, 마지막 Boot Image 영역에 boot.bin 바이너리 코드를 삽입하는 것이다. 각각의 자료형은 아래 표와 같다. 그 정보에 대한 설명은 ISO9660과 EMCA 119, 그리고 "El Torito Bootable CD-ROM Format Specification Version 1.0 January 25, 1995" 에 기술되어 있다.

4. Define Structures To Create ISO Image

```
typedef struct tagPrimaryVolumeDescriptor
{
        unsigned char volumeDescriptor;
        unsigned char standardID[5];
        unsigned char volumeDescriptorVersion;
        unsigned char unused1;
        unsigned char systemID[32];
        unsigned char volumeID[32];
        unsigned char unused2[8];
        unsigned char volumeSpaceSize[8];
        unsigned char unused3[32];
        unsigned char volumeSetSize[4];
        unsigned char volumeSequenceNumber[4];
        unsigned char logicalBlockSize[4];
        unsigned char pathTableSize[8];
        unsigned char locationOfOccurrenceTypeLPathTable[4];
        unsigned char locationOfOptOccurrenceTypeLPathTable[4];
        unsigned char locationOfOccurrenceTypeMPathTable[4];
        unsigned char locationOfOptOccurrenceTypeMPathTable[4];
        unsigned char directoryRecordForRootDirectory[34];
        unsigned char volumeSetID[128];
        unsigned char publisherID[128];
        unsigned char dataPreparerID[128];
        unsigned char applicationID[128];
        unsigned char copyrightFileID[37];
        unsigned char abstractFileID[37];
        unsigned char bibliographicFileID[37];
        unsigned char volumeCreationDate[17];
        unsigned char volumeModificationDate[17];
        unsigned char volumeExpirationDate[17];
        unsigned char volumeEffectiveDate[17];
        unsigned char fileStructureVerion;
        unsigned char reserved1;
        unsigned char applicationUse[512];
        unsigned char reserved2[653];
```

```
} PRIMARY_VOLUME_DESCRIPTOR;

typedef struct tagBootRecordVolumeDescriptor
{
        unsigned char bootRecordID;
        unsigned char standardID[5];
        unsigned char version;
        unsigned char bootSystemID[32];
        unsigned char unused1[32];
        unsigned char locationOfFirstSectorOfBootCatalog[4];
        unsigned char unused2[1974];
} BOOTRECORD_VOLUME_DESCRIPTOR;

typedef struct tagTerminatorVolumeDescriptor
{
        unsigned char bootRecordID;
        unsigned char standardID[5];
        unsigned char version;
        unsigned char unused1[2042];
} TERMINATOR_VOLUME_DESCRIPTION;

typedef struct tagBootCatalog{
        unsigned char validation_headerID;
        unsigned char validation_platformID;
        unsigned char validation_reserved1[2];
        unsigned char validation_IDString[24];
        unsigned char validation_checksum[2];
        unsigned char validation_keyByte1;
        unsigned char validation_keyByte2;
        unsigned char initEntry_bootIdicator;
        unsigned char initEntry_bootMediaType;
        unsigned char initEntry_loadSegment[2];
        unsigned char initEntry_systemType;
        unsigned char initEntry_unused;
        unsigned char initEntry_sectorCount[2];
        unsigned char initEntry_loadRBA[4];
        unsigned char optional[1984];
} BOOT_CATALOG;

typedef struct tagRootDirectoryDesc
{
        unsigned char root1_LengthOfDirectoryRecord;
```

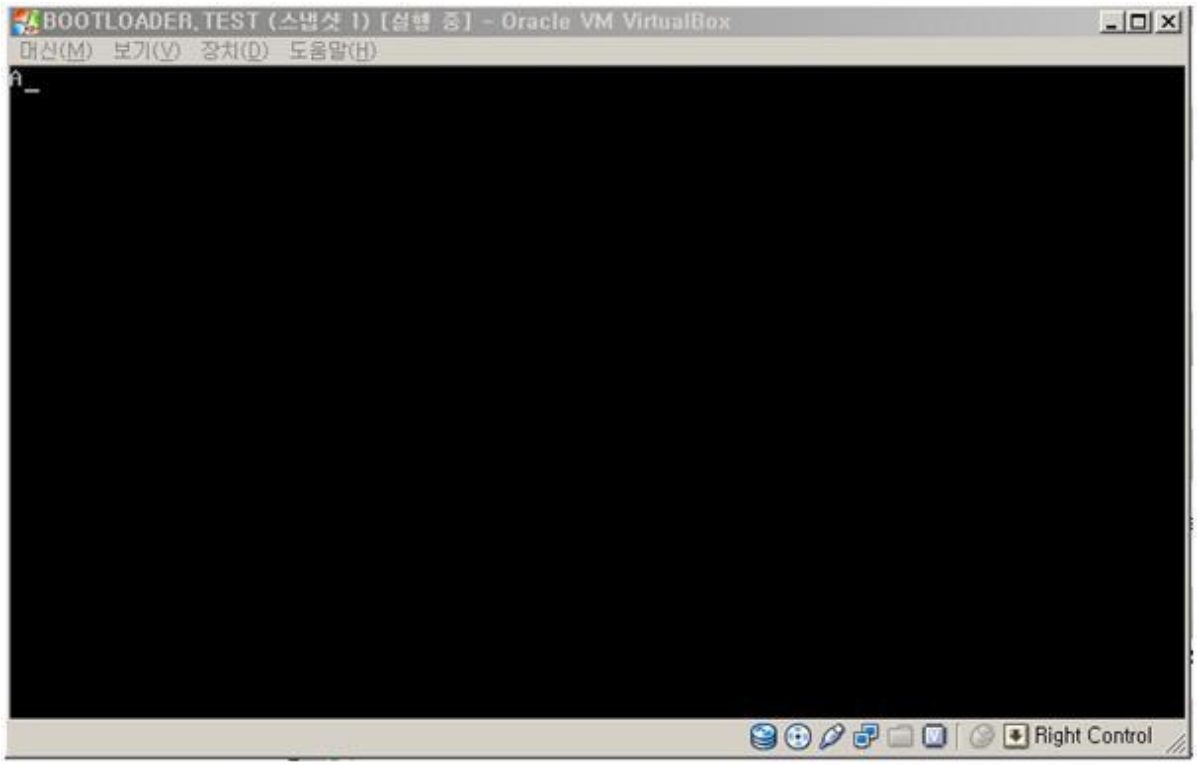
```
        unsigned char root1_extendedAttributeRecordLength;
        unsigned char root1_locationOfExtent[8];
        unsigned char root1_dataLength[8];
        unsigned char root1_recordingDateAndTime[7];
        unsigned char root1_fileFlags;
        unsigned char root1_fileUnitSize;
        unsigned char root1_interleaveGapSize;
        unsigned char root1_volumeSequenceNumber[4];
        unsigned char root1_lengthOfFileID;
        unsigned char root1_fileID;
        unsigned char root2_LengthOfDirectoryRecord;
        unsigned char root2_extendedAttributeRecordLength;
        unsigned char root2_locationOfExtent[8];
        unsigned char root2_dataLength[8];
        unsigned char root2_recordingDateAndTime[7];
        unsigned char root2_fileFlags;
        unsigned char root2_fileUnitSize;
        unsigned char root2_interleaveGapSize;
        unsigned char root2_volumeSequenceNumber[4];
        unsigned char root2_lengthOfFileID;
        unsigned char root2_fileID;
        unsigned char default_unused[1980];
} ROOT_DIRECTORY_DESC;

typedef struct tagPathTable{
        unsigned char lengthOfDirID;
        unsigned char extendedAttributeRecordLength;
        unsigned char locationOfExtent[4];
        unsigned char parentDirectoryNumber[2];
        unsigned char reserved[2040];
} PATH_TALBE;

typedef struct tagBootImage{
        unsigned char binary[510];
        unsigned char checkBytes[2];
        unsigned char reserved[1536];
} BOOT_IMAGE;
```

위의 정확한 설명은, ISO9660, EMCA 119, 그리고, El Torito Bootable CD-ROM Format Specification Version 1.0 January 25, 1995" 을 확인하면 된다.

5. TEST IN VIRTUAL BOX



정상적으로 작동하였다.

image.hex.control.iso description

파일 크기	49,152 Bytes
파일 시스템	CDFS
사용 가능한 공간	0
전체 크기	48.0KB
볼륨 이름	SIIK BOOTABLE CD

6. Build & Anaysis boot.print.iticworld.bin

6.1. NASM Source Code

```
SubjectString db 'ITICWORLD', 0
```

http://www.viralpatel.net/taj/tutorial/hello_world_bootloader.php에 기술된 소스를 이용해서, 다음과 같은 부분만을 수정하여 bin 파일을 만들었다.

6.2. Hex Value

```
BE 1E 7C E8 0B 00 EB FE B4 0E B7 00 B3 07 CD 10 C3 8A 04 46 08 C0 74 05 E8 ED FF EB F4 C3 49 54 49
43 57 4F 52 4C 44
```

6.3. Create New ISO Image File

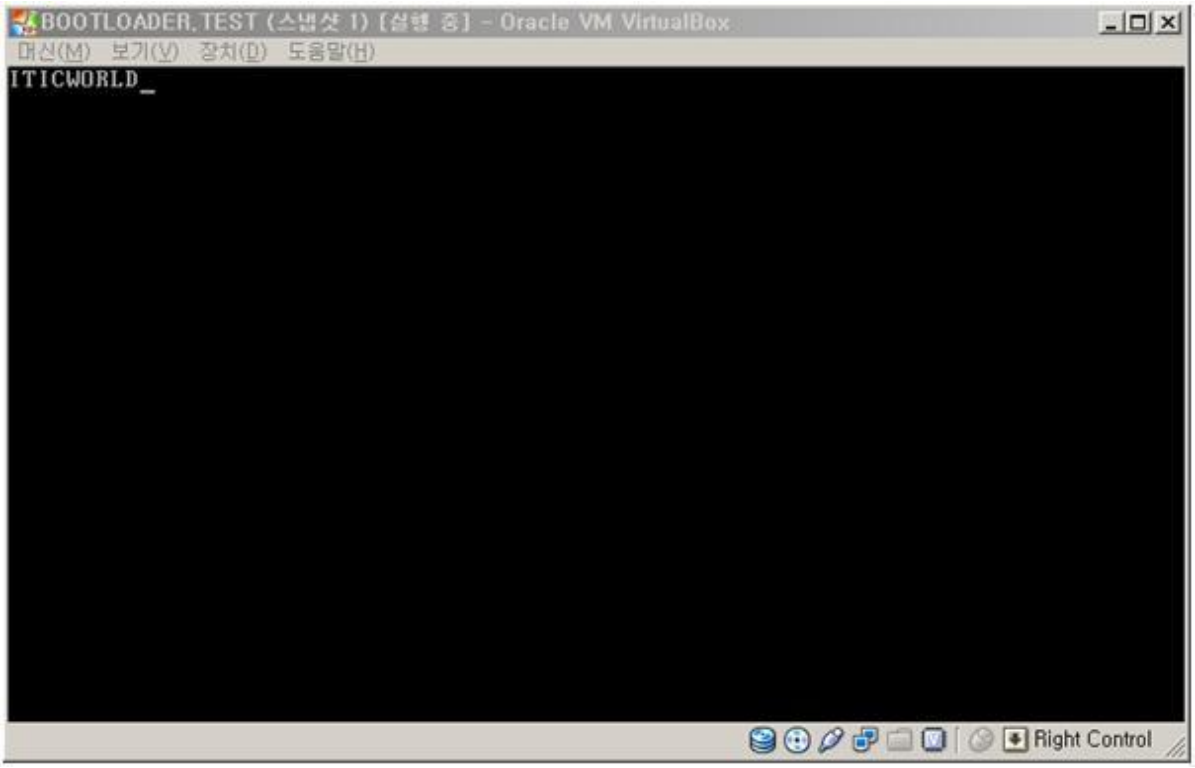
```
#ifdef MAKE_BOOTIMAGe_PRINT_A
```

```
                                {0xB0,0x41,0xE8,0x02,0x00,0xEB,0xFE,0xB4,0x0E,0xB7,0x00,0xB3,0x07,0xCD,0x10,0xC3,0x00,},
#else

{0xBE,0x1E,0x7C,0xE8,0x0B,0x00,0xEB,0xFE,0xB4,0x0E,0xB7,0x00,0xB3,0x07,0xCD,0x10,0xC3,0x8A,0x04,0x46,0x08,0xC0,0x74,0x05,0xE8,0
#endif // MAKE_BOOTIMAGe_PRINT_A
```

코드를 수정하여 실행하였다.

6.4. Result



6.5. BINARY CODE & ASSEMBLY CODE

OPCODE	DESCRIPTION
BE	MOV eSI
1E 7C	0x7C1E, (SubjectString Label) in loaded segment
E8	CALL
0B 00	0x000B
EB	Jump short
FE	ENDLESS LOOP
B4	MOV AH
0E	VALUE 0x0E
B7	MOV BH
00	VALUE 0x00
B3	MOV BL

07	VALUE
CD	INT Ib Ib: Immediate data is encoded in the subsequent byte of the instruction.
10	0x10
C3	RETN
8A	MOV Gb, Eb
04	Gb, Eb (MOD:REG:R/M)=00:000:100 Table 2-1. 16-Bit Addressing Forms with the ModR/M Byte MOV AL, [SI]
46	INC eSI
08	OR Eb, Gb
C0	Eb, Gb (MOD:REG:R/M) = 11:000:000 OR AL, AL
74	JE SHORT
05	5
E8	CALL
ED FF	CALL CURRENT POSTION + 0xFFED(signed 0x13) : CALL 0x000B OFFSET
EB	JMP SHORT jb
F4	F4 1B + 2 □ C = 0x11 JMP 0x11 (LABEL NEXT CHARACTER)
C3	RETN
SubjectString	ITICWORLD
ZERO FILLED	
55 AA	

Conclusion

"EMCA 119" 와 "El Torito Bootable CD-ROM Format Specification Version 1.0 January 25, 1995" 에서 기술한 ISO Image File에 대한 내용을 가지고, hex code만을 이용해서 Bootable .ISO Image를 만들었다. Nero Burning ROM Software의 경우, 부팅 이미지만을 만드는데, 총 600 Sector를 잡는다. 이에 대한 이유가 있겠지만, 아직은 그와 관련한 자료를 찾지 못했다. 만든 이미지는 600 Sector가 아니라, 24 Sector다. 그리고 Sector관련 size를 24으로 설정하였다. 그래도 Daemon Tools Lite, 그리고 VirtualBox 에서 정상적으로 부팅되는 것을 확인했다. 분명, 이 부분에 대한 정보를 얻고, 어떤 것이 올바른 것인지 확인해야 한다. 그리고, ISO Image와 관련한 기타 파일 시스템 Specification을 요약할 필요가 있다. 이것을 요약하고 라이브러리화 한다면, 쉽게, Image Burning Soft Ware, 그리고 Daemon Too과 같은 Virtual Image Rom을 만들 수 있을 것이다.

또한, 마지막 목표(Hex Code Editing만을 통하여 Boot-Loader을 만들고, 그 부팅 코드를 Hex Editing만을 이용해서 ISO Image File에 삽입하는)를 보다 쉽게 접근하기 위해서, "Writing Hello World Bootloader" 의 소스를 조금 수정하여, ITICWORLD란 글

자를 Booting 순간에 출력하도록 하였고, 이의 Hex 코드를 분석해 보았다.

Reference

http://thestarman.pcministry.com/asm/2bytejumps.htm

Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference

EMCA 119

El Torito Bootable CD-ROM Format Specification Version 1.0 January 25, 1995

http://www.viralpatel.net/taj/tutorial/hello_world_bootloader.php

Resource

boot.print.iticworld.bin

image.hex.control.iso

Image.iso

DEVELOPMENT

Make Boot-Loader Only To Edit Hex

Information

Subject	Make Boot Loader Only To Edit Hex
Author	siik iticworld@hanmail.net
Version	0.1
Summary	simple draw pixel about “ITICWORLD” Test Boot-Loader
SVN	https://iticworld.net:7281/svn/MAKE.BOOT.LOADER.WITHOUT.COMPILER/Make Boot-Loader Only To Edit Hex

History

Version	Author	Date	Description
0.1	siik iticworld@hanmail.net	201202202249	문서 초기 작성

Milestones

BIOS Interrupt 정리

BIOS 관련 Summary

Booting 관련 Summary

Intel® 64 and IA-32 Architectures Software Developer’s Manual 정리

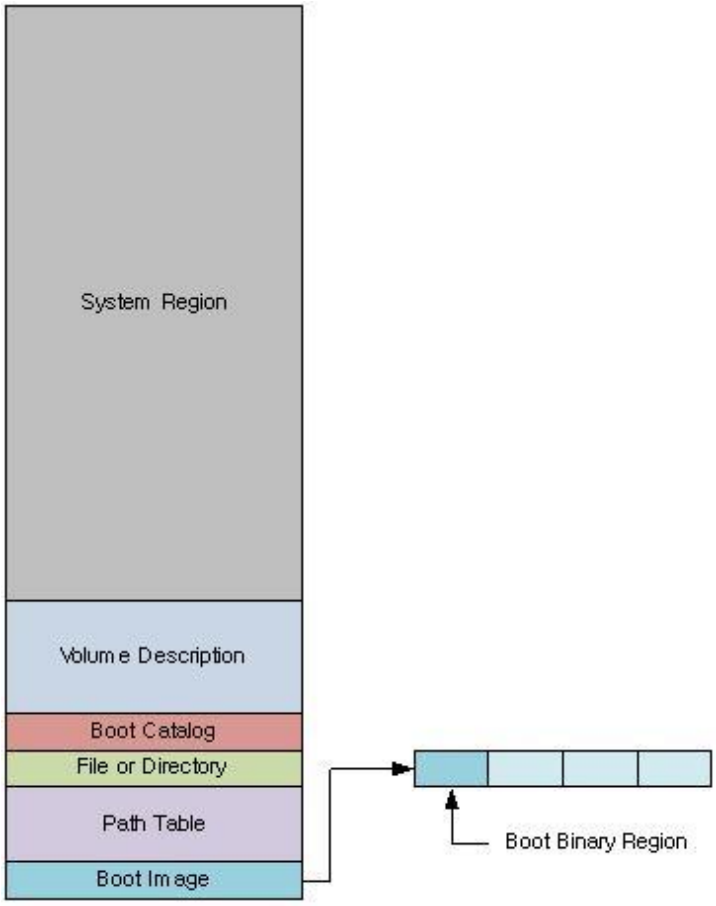
Make Executable and Linkable Format Only to edit hex

Introduction

Bootable ISO Disk Image를 Hex Code 만을 이용해서 만들고, 그것을 테스트 해보았다. 이제 그 개발 경험을 바탕으로 이제 Boot Image(Binary) 역시도, 컴파일러 없이, Specification과 Hex Code만을 가지고 만들어볼 것이다.

Description

1. Bootable ISO Disk Image Section Map



Boot Image는 Boot Catalog에서 묘사한 대로, Boot Image Sector에 저장된다. 그곳에 저장된 Boot Binary Code가 BIOS Load 후에 수행하는 코드이다. 그 코드는 16비트 어셈블리 코드로 되어 있으며, 그 크기는 512 Byte이다. 그리고, 511번째 바이트에 0x55, 그리고 512번째 바이트에 0xAA가 삽입된다. 실제 코드를 작성할 때, 로드되는 세그먼트를 고려해서 코드를 작성해야 하는데, BIOS를 통해 로드되는 세그먼트 주소는, 0x07C0다. 그렇기 때문에, Make Bootable ISO Image File to hex editing.docx 에서 설명했던 Boot Loader의 어셈블리 코드에서 MOV eSI, 0x7C1E는 실제 바이너리 코드에서는 0x001E를 참조하는 것이다.

BE	MOV eSI
1E 7C	0x7C1E, (SubjectString Label) in loaded segment

위의 맵처럼, boot binary code 는 boot image region에 첫 번째에 삽입된다. 그리고 그 크기는 512byte다. 즉, 0xB800에서 0xBA00까지 저장된다.

2. Boot Binary Assembly Code

실제로 컴파일러를 이용하지 않을 생각이다. 하지만, 구조화된 밑그림을 그리지 않고, 헥사 코드를 바로 조작하기란 쉽지 않다. 아직은 매트릭스의 주인공인 네오처럼 바로 원하는 헥사코드로 명령을 내리기란 무지 어렵기 때문이다. 그래서 일단 구조적인 모습만을 고려하고 그것을 Specification을 이용해서 헥사코드로 컴파일러 없이 변경한 후에 그 헥사코드를 이미 만들어진 ISO IMAGE에 삽입할 것이다.

#1 Basic Process

비디오 모드를 변경해야 한다.
저장된 데이터에서 X좌표를 읽는다.
만약, 값이 0이면, JMP $를 수행한다.
저장된 데이터에서 Y좌표를 읽는다.
기본 값을 설정하고, 인터럽트를 보낸다.

#2. Hex Code (Not Applied Jump Offset

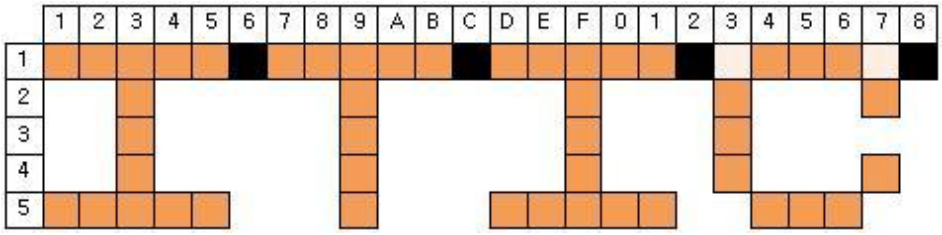
Offset	Assembly	Hex	Description
00 □ 01	MOV AH, 0x00	B4 00	Set Video Mode
02 □ 03	MOV AL, 0x12	B0 12	Graphic Mode, 16, 640x480
04 □ 05	INT 10	CD 10	Video Interrupt
06 - 08	MOV SI, ??	BE ?? ??	
09 □ 0A	MOV AH, 0x0C	B4 0C	
0B □ 0C	MOV AL, 0x02	B0 02	
0D □ 0E	MOV CH, 0x00	B5 00	
0F □ 10	MOV DH, 0x00	B6 00	
11 □ 12	MOV BH, 0x00	B7 00	
13 □ 14	MOV CL, [SI]	8A 0C	MOD:REG:R/M: 0C 00:001:100
15 □ 16	OR CL, CL	08 C9	MOD:REG:R/M:C9
17	INC SI	46	
18 □ 19	JZ ??	74 ??	Jump short
1A □ 1B	MOV DL, [SI]	8A 14	
1C	INC SI	46	
1D □ 1E	INT 0x10	CD 10	Video Interrupt
1F □ 20	JMP ??	EB ??	
21 - 22	JMP $	EB FE	
	Pixel Data		
1FE □ 1FF	Check Byte	55 AA	

#3 Calculate Jump Offset

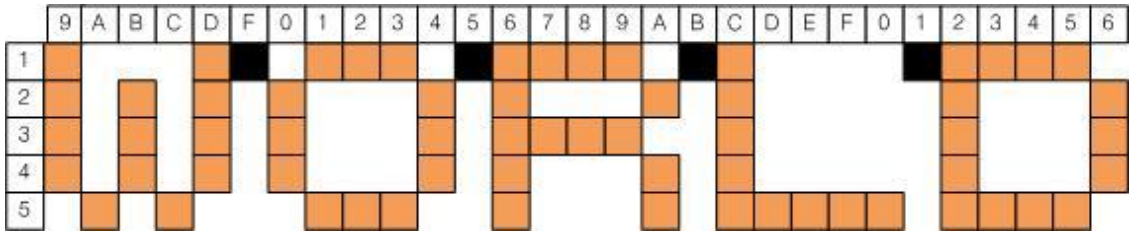
06 - 08	MOV SI, ??	BE 23 7C	Pixel Data Start Offset 0x23
18 □ 19	JZ 07	74 07	Jump short

			Offset 0x21 0x18 + 0x02 + X = 0x21 0x1A + X = 0x21 X = 0x21 □ 0x1A = 0x07
1F □ 20	JMP F2	EB F2	JUMP SHORT Offset 0x13 0x1F + 0x02 + X = 0x13 X = 0x100 + 0x13 □ 0x21 = 0xF2

#4 Insert Pixel Data



01 01 02 01 03 01 04 01 05 01 07 01 08 01 09 01 0A 01 0B 01 0D 01 0E 01 0F 01 10 01 11 01 14 01 15 01 16 01 03 02 09 02 0F 02 13 02 17 02 03 03 09 03 0F 03 13 03 03 04 09 04 0F 04 13 04 17 04 01 05 02 05 03 05 04 05 05 05 09 05 0D 05 0E 05 0F 05 10 05 11 05 14 05 15 05 16 05



19 01 1D 01 21 01 22 01 23 01 26 01 27 01 28 01 29 01 2C 01 32 01 33 01 34 01 35 01
19 02 1B 02 1D 02 20 02 24 02 26 02 2A 02 2C 02 32 02 36 02
19 03 1B 03 1D 03 20 03 24 03 26 03 27 03 28 03 29 03 2C 03 32 03 36 03
19 04 1B 04 1D 04 20 04 24 04 26 04 2A 04 2C 04 32 04 36 04
1A 05 1C 05 21 05 22 05 23 05 26 05 2A 05 0C 05 2D 05 2E 05 2F 05 30 05 32 05 33 05 34 05 35 05

#5 Merge Data

Code
Pixel Data
Zero Fill

```
Check Byte
```

이제 데이터를 병합하면,

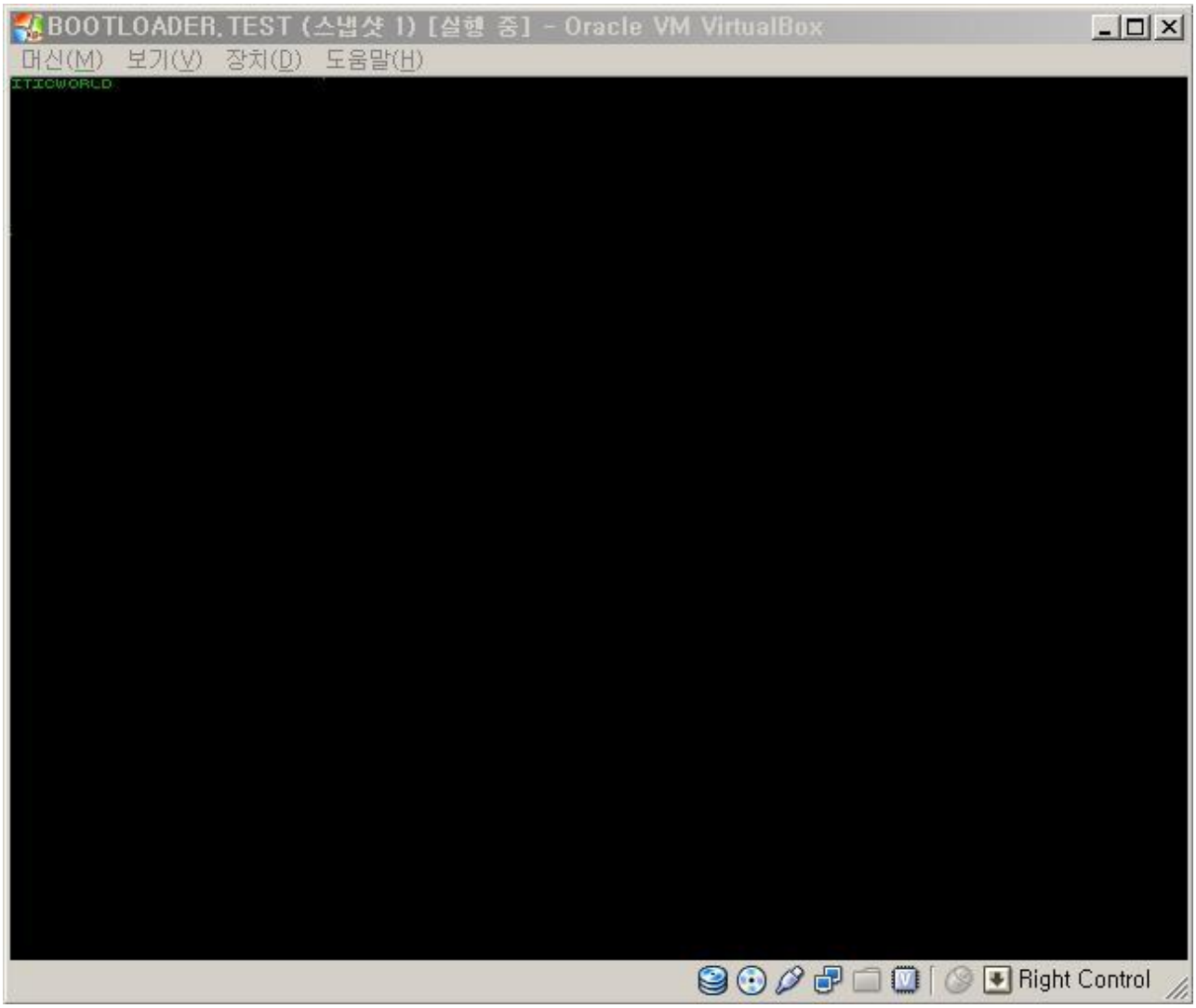
```
B4 00 B0 12 CD 10 BE 23 7C B4 0C B0 02 B5 00 B6 00 B7 00 8A 0C 08 C9 46 74 07 8A 14 46 CD 10 EB F2
EB FE 01 01 02 01 03 01 04 01 05 01 07 01 08 01 09 01 0A 01 0B 01 0D 01 0E 01 0F 01 10 01 11 01 14 01 15
01 16 01 03 02 09 02 0F 02 13 02 17 02 03 03 09 03 0F 03 13 03 03 04 09 04 0F 04 13 04 17 04 01 05 02 05 03 05
04 05 05 05 09 05 0D 05 0E 05 0F 05 10 05 11 05 14 05 15 05 16 05 19 01 1D 01 21 01 22 01 23 01 26 01 27 01
28 01 29 01 2C 01 32 01 33 01 34 01 35 01 19 02 1B 02 1D 02 20 02 24 02 26 02 2A 02 2C 02 32 02 36 02 19 03
1B 03 1D 03 20 03 24 03 26 03 27 03 28 03 29 03 2C 03 32 03 36 03 19 04 1B 04 1D 04 20 04 24 04 26 04 2A 04
2C 04 32 04 36 04 1A 05 1C 05 21 05 22 05 23 05 26 05 2A 05 0C 05 2D 05 2E 05 2F 05 30 05 32 05 33 05 34 05
35 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA
```

다음과 같이 된다. 빨간 글씨가 부트로더의 수행 코드이며, 파란 글씨가 Pixel 데이터 코드이다. 그리고 나머지는 0으로 채워지고, Boot Load check bytes(55 AA) 가 채워진다.

#6 Result

이 코드는 이미 만들어진 image.hex.control.iso의 Boot Image에 삽입했다. 이 Bootable ISO Image 파일은 손수 만들었음을 가정한다. 그 이유는 이미 그와 관련하여 손수 헥사코드만을 가지고 만들어 보았고, 그렇게 만든 파일을 가지고 작업했기 때문이다. 이와 관련하여 아래의 링크에 기술되어 있다.

https://iticworld.net:7281/svn/MAKE.BOOT.LOADER.WITHOUT.COMPILER/Make Bootable ISO Image File only editing hex code/



녹색 글씨로, "ITICWORLD"가 출력된 화면이다.

Conclusion

시간이 꽤 걸렸다. 실제로 Assembly Code를 작성하는데는, 1hour가 걸리지 않았다. 몇 분만에 코드를 작성했다. 하지만, 그렇게 밑그림을 그린 코드를 가지고, 컴파일러 없이, 헥사코드 값을 살펴가면서, 각 값을 가지고, 수작업으로 Hex code로 변화하는데, 시간이 많이 걸렸다. 약 4 Hour 정도 걸렸다. 이 안에는 잘못된 Hex Code를 수정해서 출력하는 것도 포함이 된다. 이것을 통해서 컴파일러가 얼마나 필요한 것인지를 알 수 있다. 실제, Hex Code Map 을 외우고 있지 않고, 그 변환 규칙 또한 외우고 있지 않지만, 감으로, 시간이 더 걸릴 것을 알 것 같다. 여기서는 몇 줄 안되는 Assembly 밑그림 코드여서 시간이 더 걸리지 않았지만, 만약 더 긴 코드와 그 코드를 눈으로 Validation하기란 쉽지 않고, 또한 시간이 많이 걸릴 것이란 생각이 든다. 끝으로, 조금 더 Hex 값들에 익숙해지기 위해서, 헥사 코드만을 다루어 Executable and Linkable Format(ELF)을 만들어 보는 것도 해볼 가치가 있다.

Reference

https://iticworld.net:7281/svn/MAKE.BOOT.LOADER.WITHOUT.COMPILER/Make Bootable ISO Image File only editing hex code/[DEVELOPMENT]Make Bootable ISO Image File to hex editing.docx

http://www.viralpatel.net/taj/tutorial/hello_world_bootloader.php
http://en.wikipedia.org/wiki/BIOS_Color_Attributes
Intel® 64 and IA-32 Architectures Software Developer's Manual
http://faculty.kfupm.edu.sa/EE/ahussain/teaching/Exp_06.pdf

Resource

image.hex.control.2.iso

BootLoader ISO Image File To Edit Hex

블로그 나다운 세상 http://blog.daum.net/iticworld

저자 씨익

발행일 2012.02.20 23:01:12

Daum 블로그